386 | DOS-Extender Release Notes for Version 4.1

Phar Lap Software, Inc.
60 Aberdeen Ave., Cambridge, MA 02138
(617) 661-1510
FAX (617) 876-2972
tech-support@pharlap.com

First Edition: January 1992

386 | DOS-Extender Release Notes for version 4.1

This file documents differences between the 4.1 version of 386 | DOS-Extender (RUN386.EXE) and the third edition of the *386 | DOS-Extender Reference Manual*, published in January 1991, and the second edition of the *386 | DOS-Extender Programmer's Guide to DPMI and Windows*, published in January 1992.

Windows 3.*x* Support

The *386 | DOS-Extender Programmer's Guide to DPMI and Windows* documents Windows support as of version 4.0 of the 386 | DOS-Extender. This manual describes in detail many of the issues involved with writing 386 | DOS-Extender applications which run in the DOS compatibility box of Windows 3.*x* Enhanced mode. Included in the manual is a large amount of C example code to illustrate the ideas covered in the text.

In version 4.1, support for the -REALBREAK switch, and the 250Fh call (Convert Protected-Mode Address to MS-DOS Address) has been added for Windows 3.*x*. To enable -REALBREAK support, you must install the PHARLAP.386 Windows device driver in the Windows SYSTEM.INI file, by adding the following line to the [386enh] section of the file:

```
device=pharlap.386
```

Under Windows 3.*x*, by default the EMulate bit in CR0 cannot be turned on with the 2535h system call. To make the EMulate bit available, install the PHARLAP.386 device driver provided in the SDK by placing the following line after all the "device=*" lines in your SYSTEM.INI file:

```
device=pharlap.386
```

When debugging under Windows 3.*x*, by default the debug registers are not available. This means data watchpoints cannot be used. To use data watchpoints, install the device driver provided in the SDK by placing the following line after all the "device=*" lines in your SYSTEM.INI file:

```
device=pharlap.386
```

Interrupt Handlers Under Windows 3.*x*

Windows 3.*x* Enhanced mode supports demand-paged virtual memory. Page faults in a hardware interrupt handler or a critical error handler are not allowed; if one occurs it will crash Windows.

BEFORE installing a hardware interrupt handler or critical error handler, you MUST lock all memory that is read or written by that handler. This includes all code and data accessed by the handler, and also stack memory if the handler switches stacks. It is not necessary to lock the stack if you don't switch stacks, since the stack provided when your interrupt handler is entered is always locked.

Use the 251Ah or 252Bh subfunction 5 system calls to lock memory, and the 251Bh or 252Bh subfunction 6 system calls to unlock memory. These calls are documented in the *386 | DOS-Extender Programmer's Guide to DPMI and Windows.* These are the same calls used to lock memory under Phar Lap's 386 | VMM virtual memory system. When not running under Windows or 386 | VMM, these calls do nothing and always return success. Correct procedure is therefore to always lock down hardware interrupt handlers using these calls.

Do not increase the size of a segment which is accessed by a hardware interrupt handler (for code, data, or stack). The segment may be moved in the linear address space, and under DPMI there is no way to atomically update the segment descriptor when the segment moves, so there is a window where a hardware interrupt can come in and the descriptor points to the old linear address, which is no longer valid. If you need to resize a segment which is accessed by a hardware interrupt handler, first unhook the hardware interrupt, then resize the segment, then rehook the interrupt.

If you write an interrupt handler that chains to the previous handler, you cannot forge a return address to get control back again when the previous handler IRETDs (by changing the 386 | DOS-Extender umbrella CS:EIP values in the stack frame) under Windows 3.*x*.

Windows 3.0 has a bug which prevents 32-bit programs from hooking interrupts 40h and higher. By default, 386 | DOS-Extender patches Windows to fix this bug, so any interrupt vector can be hooked normally. However, if 386 | DOS-Extender is unable to install the patch, or if the -NOPATCHWIN30 switch is used to prevent installing the patch, 386 | DOS-Extender will not permit the application to hook interrupts 40h and above. If the patch is not installed, system calls 2504h and 2507h for vectors 40h and above will return an error (carry flag set). System call 2506h (pass interrupt to protected mode) will succeed, but your handler will always get control from real mode (i.e., even if the interrupt occurs in protected mode, 386 | DOS-Extender will let Windows pass it down to real mode, and 386 | DOS-Extender will then pass it up to protected mode and invoke your handler).

Memory Preallocation Under Windows 3.0

Windows 3.0 has a bug that prevents growing a segment (data is lost under certain conditions when segment size is increased). This bug is fixed in Windows 3.1, and the workarounds described here are deactivated (and do not have any bad side effects) when running in any environment other than Windows 3.0 Enhanced mode.

386 | DOS-Extender has a two-part strategy for working around this bug:

1. **Pre-allocation**
   The main program segment (both for the main application program, and for any programs loaded later with the Load EXP File (2529h) or Load for Debug (252Ah) calls) gets additional memory preallocated when the program is loaded. The maximum size of the program segment can then always grow as large as the preallocated size.

2. **-WIN30GROW**
   By default, attempts to grow the program segment beyond its preallocated limit (or to grow a segment allocated with INT 21h func 48h at all) will be failed under Windows 3.0. If the -WIN30GROW switch is specified, then 386 | DOS-X will check to see if growing the segment will cause it to move (which is the bug condition); if not, it will grow the segment. (This switch is NOT the default because the behavior will change depending on whether other applications are allocating memory in the system. If you are testing your program as the only active application, then growing the program segment will almost always succeed; but when an end-user runs the application and then opens another window and fires up another program, suddenly your program will stop being able to allocate memory because growing the program segment would cause it to move. So before deciding to use -WIN30GROW, think about the implications).

The only thing the application developer can affect (other than deciding whether or not to use the -WIN30GROW switch) is how large the preallocated program segment size is. The default if no switches are used is 2 MB more than the minimum program load-time memory needs. (Note that at load time, the program's segment limit is set to its minimum size, not to the preallocated size, and its limit is increased as usual as requests to grow the segment are made. The preallocated memory beyond the initial segment limit is not initialized to zero until the segment limit is increased, to avoid unnecessary thrashing at program load time.)

When deciding how much memory should be preallocated, the tradeoff is how much heap memory your program needs at run-time to run efficiently, versus consuming so much virtual memory that Windows doesn't have enough left to run other programs. Remember that Win 3.0 limits total virtual memory to four times the physical memory on the machine, regardless of available disk space. At load time, 386 | DOS-X will preallocate as much memory as possible for your program, up to the selected preallocation size (in other words, if there's enough memory to load your program but not to satisfy the desired preallocation size, the program still loads successfully).

There are two ways to control the preallocation size under Win 3.0: the link-time -MAXDATA switch, and the run-time -WIN30LIMIT switch. For example to specify that an additional 3 MB of memory be preallocated for your program segment at load time, either link with

```
-MAXDATA:        386link myprog -maxdata 300000h
```

or run with

```
-WIN30LIMIT:     run386 -win30limit +300000h myprog
```

The -MAXDATA switch is preferable for two reasons: (1) you can specify it at link time, so it's more convenient, and (2) if your program loads other EXP files with the 2529h system call, using -MAXDATA allows the preallocation size to be application-specific for each EXP file, but the -WIN30LIMIT switch is applied globally to all loaded EXP files. Using the -MAXDATA switch has no bad side effects in other environments, because by default all extra load-time memory is immediately released by the compiler run-time initializer when your program starts up, and then is incrementally allocated again as your program requests memory.

To summarize, the best thing to do to make sure your program gets adequate amounts of memory under Windows 3.0 without hogging memory it doesn't need, is to figure out the MINIMUM amount of heap memory your program needs to run ADEQUATELY, and then link your program with -MAXDATA n. This will make sure your program runs reasonably well under Windows 3.0, and will not affect program operation in any other environment. (Note under ANY environment, INCLUDING Win 3.0, you should never ASSUME you will get a minimum amount of heap memory, because your program will still load and run successfully if there is enough memory to satisfy the minimum load requirements but not the entire preallocation size).

New Switches

-XMS AUTO

The default setting of the XMS switch, this means assume that the XMS lock block call returns linear addresses under 386-to-the-Max, physical addresses under all other XMS drivers. Since the XMS spec is interpreted ambiguously in different XMS implementations, 386 | DOS-Extender needs to make assumptions (or be told) whether the XMS lock block call returns physical or linear addresses.

-XMS OFF

Don't use any XMS memory

-XMS PHYS[ICAL]

Assume the XMS lock block call returns physical addresses

-XMS LIN[EAR]

Assume the XMS lock block call returns linear addresses

-NEST[DPMI]

Assume the DPMI host supports multiple client programs

-NONEST[DPMI]

Assume the DPMI host does NOT support multiple client programs. By default, 386 | DOS-Extender assumes that any DPMI version 0.9 host does not support multiple clients, and will refuse to run if another DPMI client is detected. Windows 3.0 and 3.1 definitely do NOT support multiple clients, so don't use the -NESTDPMI switch under Windows.

-NOPCD[WEITEK]

Don't assert the PCD (Page Cache Disable) bits for Weitek or Cyrix memory-mapped coprocessors on 486 or later PCs. By default, the PCD bits (which only exist on the 486 or later) are asserted to disable any caching of memory in the memory-mapped address range used by the coprocessor. On Compaq machines, the PCD bits are NOT asserted to work around a hardware bug on Compaq 486/33L machines. This switch can be used to turn off PCD bits on any 486 or later PC.

The following switches were added to control 386 | DOS-Extender's use of DPMI services and are described in detail in the *386 | DOS-Extender Programmer's Guide to DPMI and Windows*:

-MAXD[ATADPMI] nbytes	-DPMI[SIM] 0.9
-WIN30L[IMIT] [+]nbytes	-DPMI[SIM] 1.0
-WIN30D[BG]L[IMIT] [+]nbytes	-NODPMI
-WIN30GROW	

Modified Switches

-WIN30L[IMIT] [+]nbytes

In the 4.0 version of 386 | DOS-Extender, the additional memory requested with this switch was "all or nothing"; if less than the requested amount was available, 0 additional bytes of memory were preallocated. In this release, as much memory as possible up to the amount requested is preallocated. For example, suppose your program requires 3 MB of memory to load, and you configure in -WIN30L +200000h to request an additional 2 MB of preallocated memory under Windows 3.0. On a system with only 4 MB of memory available, the 4.1 386 | DOS-Extender allocates all 4 MB for your program segment, but the 4.0 386 | DOS-Extender only allocates the minimum 3 MB amount. (Note that using the -MAXDATA switch at link time is now the recommended workaround for the Windows 3.0 segment growing bug; see discussion above under "Windows 3.*x* Support".)

-REALBREAK

Now supported under Windows 3.*x*, if the PHARLAP.386 Windows virtual device driver is installed.

New System Calls

INT 21h — **Read/Write IDT Descriptor AX = 253Dh**
ver: 4.0

```
In:     BL = interrupt number
        ECX = 0, if reading descriptor contents
              1, if writing descriptor contents
        DS:EDX = pointer to 8-byte buffer to hold descriptor
```

```
Out:    If success:
              Carry flag = clear
              If reading descriptor, buffer at DS:EDX filled in
        If failure:
              Carry flag = set
              EAX     = error code
                      = 130, if running under DPMI
```

This system call is used to read and write hardware-level IDT descriptors directly, rather than reading and writing the internal 386 | DOS-Extender "shadow" IDT (see section 6.2 of the reference manual).

No checking of descriptor contents is done when writing the descriptor. Be sure to set the Descriptor Privilege Level (DPL) bits in the access rights byte of the descriptor equal to the Current Privilege Level (CPL), which can be obtained from the two LSBs of the CS register, or by making the Get Configuration Information system call (2526h).

By installing handlers that are vectored to directly through the IDT, you can avoid the overhead of the 386 | DOS-Extender "umbrella" interrupt handler that sets up the standard interrupt stack frame documented in section 6.5 of the manual.

Normally, this call can only be used by programs that run at privilege level zero. The reason is the processor does not allow direct vectoring to a less privileged level when an interrupt occurs. However, direct vectoring is permissible if you know the interrupt can only occur while the application program is executing. An example of this is the INT 7, the Coprocessor Not Available exception that occurs when a 287/387 instruction is executed and the EM (emulate) bit of CR0 is set. 386 | DOS-Extender never executes any coprocessor instructions in protected mode, so INT 7 is guaranteed to only occur when running at the application privilege level. Direct vectoring to an INT 7 handler can significantly enhance the performance of 387 emulation software.

It is never possible to directly access the IDT under any DPMI host, so applications that use this call must be prepared for an error return if DPMI compatibility is desired.

INT 21h **Read Page Table Entry and Page Table Info**
AX = 252Bh, BH = 9 **ver: 4.1**

In: If BL = 0:
    ECX = linear addr to read PTE for
  If BL = 1:
    ES:ECX = addr to read PTE for

Out: If success:
    Carry flag = clear
    EAX = page table entry
    EBX = additional page table information
  If failure: Carry flag = set
    EAX = 9, if invalid address
      = 130, if running under DPMI

INT 21h **Write Page Table Entry and Page Table Info**
AX = 252Bh, BH = 10 **ver: 4.1**

In: If BL = 0:
    ECX = linear addr to write PTE for
  If BL = 1:
    ES:ECX = addr to write PTE for
  ESI = page table entry
  EDI = additional page table information

Out: If success:
    Carry flag = clear
  If failure:
    Carry flag = set
    EAX = 9, if invalid address
      = 130, if running under DPMI

386 | DOS-Extender keeps an additional DWORD of information that supplements the page table entry for each page. If using the read and

write page table entry calls to copy page information (NOT RECOMMENDED), be sure to copy the additional page information as well as the PTE.

INT 21h **Map Data File at File Offset**
AX = 252Bh, BH = 11 **ver: 4.1**

```
In:     If BL = 0:
           ECX = linear addr to map data file at
        If BL = 1:
           ES:ECX = addr to map data file at
        EDX = number of bytes to map
        DS:EDI = pointer to zero-terminated file name string
        DS:ESI = pointer to data structure, described below
```

```
Out:    If success:
           Carry flag = clear
        If failure:
           Carry flag = set
           EAX = 2, if file error
             = 9, if invalid address
             = 129, if invalid parameters, or 386|VMM not present
             = 134, if maximum number of 386|VMM file handles (4)
                already in use
           If EAX = 2 (file error)
                ECX = 1, if DOS error opening file
                   = 2, if DOS error seeking in file
                   = 3, if DOS error reading from file
                EDX = error code returned by DOS
```

This call maps a data file into the program's virtual address space. It is equivalent to 252Bh subfunction 3, except that this call allows a starting offset in the file to be specified, while subfunction 3 always maps the file starting at file offset 0. This call is only available under 386|VMM, and is not supported when running under a DPMI host.

The data structure at DS:ESI contains the following:

0000h	DWORD	starting file offset to map
0004h	DWORD	DOS file access and sharing mode, as defined for INT 21h function 3Dh, Open File

Modified System Calls

INT 21h AH = 48h — Allocate Segment

In: As documented in manual

Out: As documented in manual

When 386 | DOS-Extender searches the LDT for a free descriptor to allocate, it now considers any nonzero descriptor to be allocated. Previously, only descriptors with the present bit set were considered allocated. This change allows applications to allocate LDT descriptors and then mark them not present, provided at least one other bit in the descriptor is set.

INT 21h AX = 250Fh — Convert Protected-Mode Address to MS-DOS Address

In: As documented in manual

Out: As documented in manual

This call is now supported for -REALBREAK memory under Windows 3.*x*, if the PHARLAP.386 Windows virtual device driver is loaded.

INT 21h AX = 251Eh — Exchange Two Page Table Entries

In:	As documented in manual
Out:	As documented in manual

The exchange page table entries call now also exchanges the additional DWORD of information 386 | DOS-Extender maintains on each page.

INT 21h AX = 2520h — Get Memory Statistics

In:	As documented in manual
Out:	As documented in manual

The DWORD at offset 5Ch in the buffer now contains a count, in pages, of the largest available free block of memory. When running without 386 | VMM, this is just the number of free physical pages (the number returned by the INT 21h AH = 48h allocate memory call when it fails). Under 386 | VMM, this count also includes available disk space for virtual memory. Under DPMI, this is the largest free memory block reported available by the DPMI host.

INT 21h AX = 2526h — Get Configuration Information

In:	As documented in manual
Out:	As documented in manual

There are four new flag bits defined in Flags DWORD 1 (at offset 0000h in the configuration buffer). They are:

01000000h	set if -NESTDPMI switch used
02000000h	set if -NONESTDPMI switch used

04000000h	set if -NODPMI switch used
08000000h	set if -NOPCDWEITEK switch used

INT 21h AX = 2529h — Load Flat Model .EXP or .REX File

In: As documented in manual

Out: As documented in manual

An additional parameter is returned in the parameter block at ES:EBX. At offset 001Ch a flags doubleword is returned. Bit zero of the flags doubleword is set if the child was linked with the -UNPRIVILEGED switch, cleared if the child was linked with -PRIVILEGED. All other bits are reserved and are always cleared.

INT 21h AX = 2534h — Get Interrupt Flag

In: As documented in manual

Out: As documented in manual

Fixed a bug that existed in version 4.0: the interrupt flag was not correctly returned under a DPMI host such as Windows 3.*x*.

INT 21h AX = 2535h — Read/Write System Registers

In: As documented in manual

Out: As documented in manual

By default, these registers are not available under Windows 3.*x*. To make them available through this system call, you must install the PHARLAP.386 virtual device driver as described in the *386 | DOS-Extender Programmer's Guide to DPMI and Windows.*

INT 21h AX = 253Ch — **Shrink 386 | VMM Swap File**

In: As documented in manual

Out: As documented in manual

This call now refuses to shrink the swap file smaller than the minimum swap file size specified with the -MINSWFSIZE switch. Also fixed a bug which sometimes resulted in the swap file size being INCREASED by this call.

386 | DOS-Extender Detection

386 | DOS-Extender now provides real mode INT 2Fh calls which can be used to determine if 386 | DOS-Extender is present in memory.

INT 2Fh AX = ED00h — **386 | DOS-Extender detection**

In: BL = 3 if requesting SDK version of 386 | DOS-Extender
BL = 4 if requesting RTK version of 386 | DOS-Extender

Out: If requested version of 386 | DOS-Extender present:

```
        AX = EDFFh
        SI = 5048h ('PH')
        DI = 4152h ('AR')
        CH = 386 | DOS-Extender major version number
        CL = 386 | DOS-Extender minor version number
        DX = flags
           bit 0001h set if running under DPMI
           bit 0002h set if 386 | VMM present
        If running under DPMI:
                BH = DPMI major version number
                BL = DPMI minor version number
If 386 | DOS-Extender not present:
        AX = ED00h
```

INT 2Fh AX = F100h — **Universal DOS Extender detection**

In: None

Out: If a DOS extender present:
    AX = F1FFh
    SI = 444Fh ('DO')
    DI = 5358h ('SX')
  If no DOS extender present:
    AX = F100h

INT 2Fh AX = ED03h — **Get 386 | DOS-Extender Entry Point**

In: CX = real mode paragraph addr of code segment
  DX = real mode paragraph addr of data segment

Out: If success:
    Carry flag = clear
    CX = protected mode code selector
    DX = protected mode data selector
    ES:DI = real mode address of 386 | DOS-Extender
      routine to use to call a protected mode function
      (see the documentation for the 250Dh system call)
  If failure:
    Carry flag = set
    AX = 8, if unable to allocate LDT descriptors

This call may only be made after determining that 386 | DOS-Extender is present with the ED00h call. This call may only be made from real mode.

The LDT descriptors allocated for the code and data segments are initialized with a base equal to the linear address of the segment in the first megabyte, a limit of 64K, and the USE32 attribute set.

New EMS Emulator Support

The "noems" ("ems=0" for 386MAX) option is now supported for the Microsoft EMM386, Quarterdeck QEMM, Qualitas 386MAX, and Compaq CEMM EMS emulators.

The "vs:y" option on Quarterdeck QEMM version 6.0 and later is now supported.

STUB386 Stub Loader Utility Program

The STUB386.EXE stub loader program included on the distribution disks can be used to avoid having to type "run386" each time you run your application program.

Bind the STUB386.EXE file to the application .EXP file to create an .EXE file that can be run by typing the file name, just like any other DOS program. This can be done with the MS-DOS COPY command. For example, to create a HELLO.EXE file by binding STUB386.EXE and HELLO.EXP:

```
copy/b stub386.exe+hello.exp hello.exe
```

386 | LINK can automatically bind STUB386 to create an .EXE output file if STUB386.EXE is given as one of the input object files in the link string. For example, to link HELLO.OBJ, and create an output .EXE file with the stub loader bound on the front:

```
386link hello stub386.exe
```

An .EXE file with the stub loader bound into it can be run by typing the file name directly:

```
hello
```

If you need to specify 386 | DOS-Extender switches for a program bound with the stub loader, you can configure them directly into the bound .EXE file with the CFIG386 utility program. For example, to limit the XMS memory usage of the bound program HELLO.EXE to one megabyte:

```
cfig386 hello -maxxmsmem 100000h
```

When you type the name of the .EXE file to run the program, DOS loads the stub loader bound to the front of the file. The stub loader searches the execution PATH for the RUN386.EXE file and loads it into memory. RUN386 then loads the application program from the bound file. There is no extra memory consumed by the stub loader program; it is completely removed from memory after RUN386 is loaded. If you rename your RUN386.EXE file, you can specify the new name to the stub loader with the CFIG386 -DOSXNAME switch documented in section 1.8 of the CFIG386 Utility Guide.

The stub loader cannot be used with 386 | DOS-Extender versions prior to 3.0 when bound to the application program as described above. However, it can be used with any 386 | DOS-Extender version by copying it to a file with the same name as the application .EXP file, but with an .EXE filename extension. When used in this way, approximately 1500-2000 bytes of conventional memory are consumed by the stub loader.

Changes to Include Files

The PHARLAP.H file in the INCLUDES directory has been split into two files: PHARLAP.H and PLDOS32.H. The file PLDOS32.H has definitions for the C callable functions available in the library DOS32.LIB. These functions all begin with the prefix _dos_. The reason for the split was to eliminate conflicts between functions with the same name defined in the PHARLAP.H and the MetaWare include file DOS.H. If you are going to be using the _dos_xxxx functions from DOS32.LIB, you should include PLDOS32.H. If you are going to be using the _dos_xxxx functions included with the MetaWare compiler, you should include DOS.H.

New Example Code

The EXAMPLES\DPMI and EXAMPLES\REALCOPY directories contain the example programs from the *386 | DOS-Extender Programmer's Guide to DPMI and Windows* manual.

The EXAMPLES\LOADER directory on the distribution disks contains a program, LOAD.C, that demonstrates the use of the Load EXP File system call (2529h). It also includes a sample program (FARHELLO.ASM) that RETurns to the caller on completion rather than making a DOS program terminate call; this is required for programs loaded with the Load EXP File system call. The LOAD.EXP program takes the first argument on the command line as the name of a file to load and execute; to load and run FARHELLO.EXP, type:

```
run386 load farhello.exp
```

Changes to Appendix F (Writing DPMI-Compatible Applications)

Appendix F of the *386 | DOS-Extender Reference Manual* has been subsumed by the new manual entitled *386 | DOS-Extender Programmer's Guide to DPMI and Windows.* For any discrepancies between the two sources, the new manual should be considered correct.

For those readers already familiar with Appendix F, the major differences are as follows:

-OFFSET is permitted under all DPMI hosts. If the host DPMI implementation does not include Paging Support (as in Windows 3.*x*), the offset pages at the beginning of the program segment are actually allocated and will not cause page faults if referenced.

The -PRIVILEGED and -UNPRIVILEGED switches are now ignored when running under DPMI, and the application is always run unprivileged, regardless of how it was linked. If you don't want your program to run in DPMI environments, configure in the -NODPMI switch.

Under DPMI, the default value for the -MAXREAL switch is FFFFh (leave as much conventional memory free as possible), rather than 0. This is because under DPMI conventional memory cannot be not used for the application program, so by default it is left free for other programs. However, if you know the conventional memory will not be needed for other programs, you can obtain better performance by explicitly specifying -MAXREAL 0; 386 | DOS-Extender will then allocate all available conventional memory and mark it pageable, thus freeing up the physical memory allocated for that DOS box to be used by your application, reducing the likelihood of virtual memory thrashing.

It is allowed for one 386 | DOS-Extender program to EXEC to another 386 | DOS-Extender program under any DPMI host. However, it is not possible for the second 386 | DOS-Extender program to call back to the first 386 | DOS-Extender program through real mode code, as it is in other environments. Also, 386 | DOS-Extender will refuse to run if any DOS extender other than 386 | DOS-Extender is already running in that DOS box, because DPMI 0.9 does not support multiple DOS extenders in one DOS box. The -NESTDPMI switch can be used to force 386 | DOS-Extender to run even if another DOS extender is already present; but under Windows 3.0 or 3.1 using -NESTDPMI will cause a crash because Windows does not support multiple DPMI clients.

It is important to keep in mind that most DPMI hosts (including Windows 3.*x* Enhanced mode) provide a virtual memory environment. This means that when INT 21h func 48h (allocate segment) returns an error, the returned size of available memory includes all available disk space. Don't try to allocate this amount unless you really need it; it will take a long time for Windows to allocate it, and it will consume all the free disk space. Under Windows 3.*x*, the returned number should not be considered accurate since Windows does not take available disk space into account when returning the amount of available virtual memory.

Miscellaneous Manual Changes and Errata

Section 3.8 of the 386 | DOS-Extender Reference Manual documents the amount of conventional memory needed to EXEC to a second copy of 386 | DOS-Extender as 65K. That is actually the minimum required run-time memory for 386 | DOS-Extender; 150K of conventional memory must be available for 386 | DOS-Extender to load and initialize.